# C13

Thu Dec 27 11:23:05 2007 Function Index Page 2

Thu Dec 27 11:23:05 2007 Function Index Page 1

Thu Dec 27 11:23:05 2007 Function Index Page 2

```
/*
** Copyright 1996,1997 EMC Corporation
*/

/*
** EDMDispatch.c
**
** Mission Statement: This is the main service file for the dispatch daemon.
**                    This file contains the callbacks from the main function
**                    which prepares the daemon to go off and service RPC's.
**
** Primary Data Acted On:
**
** Compile-Time Options:
**
**                      None.
**
** Basic idea here: Module for UNIX specific daemon initialization
*/

/*
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/
#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMDispatch.c,v $ "
                            "$Revision: 1.23 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

/* #define _POSIX_SOURCE   unable to compile with this define set */
/* #define _XOPEN_SOURCE   unable to compile with this define set */

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <stdarg.h>
#include <string.h>
#include <syslog.h>
#include <pthread.h>
#include <thread.h>
#include <sys/resource.h>

#include <logging/logging.h>
#include <util/esl_core.h>
#include <util/esl_pidfile.h>
#include <util/esl_daemon.h>
#include <csc/cscomm.h>

#include <restore/csc_EDMDispatch.h>

#include <EDMmain.h>
#include <EDMDD_ccw.h>
#include <EDMDispatchLog.h>
#include <EDMDispatchBackground.h>
#include <EDMDDcr_rstsvc.h>

/*
 * Need to define _XOPEN_SOURCE for signal funtion definitions
 * and certain signal structure definitions.
 */
#define _XOPEN_SOURCE
```





```
#include <signal.h>

#undef _XOPEN_SOURCE

static rpc_if_handle_t if_spec;

static int       G_debug = FALSE;
                                /* Variable which will disable forking */

static char      **commandlineargs; /* Pointer to command line args */

/*****************************************************************************
 ********
**
** Routine: IsDebugOn
**
** Inputs: None
**
** Outputs:  None
**
** Return Codes:
**              TRUE if debug is on.
**
** Purpose:     This routine can be used to tell other subsystems
**              whether debugging is available.
**
** Intended caller:  internal only.
**
*****************************************************************************
 ********
*/
boolean_ty
IsDebugOn()
{
    return G_debug;
}
```

```
99  /*****************************************************************************
                                                                          ********
100 **
101 ** Routine: kill_handler
102 **
103 ** Inputs: int sigval - the signal which was receieved.
104 **
105 ** Outputs:  Will log messages telling what action is being taken.
106 **
107 ** Return Codes:
108 **               exits with the number of the signal received
109 **
110 ** Purpose:     This routine handles specific signals i.e. SIGINT,
111 **              SIGQUIT,
                          SIGTERM. Each results in a log entry and an exit.
112 **
113 ** Intended caller:  internal only.
114 **
115 *****************************************************************************
                                                                          ********
116 */
117 static void kill_handler( IN int sigval )
118 1 {
119 1 int      status;
120 1 time_t   current_time;
121 1 char     *ctimebuf;
122 1 char     *ebuff;

124 1   /* If main exits, it calls this routine with signal 0 */

126 1   /* Unregister the interface */
127 1   (void) csc_unregister_server_interface(&if_spec, &status);

129 1   /* If the unregister fails, report the problem, but continue */
130 1   if ( status != error_status_ok )
131 2   {
132 2     ebuff = (char *) csc_get_error( status );

134 2     (void) EDMDispatch_logent(
                        __FILE__,__LINE__, LOG_ERR, MESSAGE_NO_LOGIN, 0,
135 2                      "CSC_SERVER_LOGIN failed: <%d> %s",
136 2                      status, (ebuff ? ebuff : "Unknown error") );
137 1   }

139 1   /* Get the current time */
140 1   (void) time(&current_time);

142 1   ctimebuf = ctime(&current_time);

144 1   /* Overlay newline with null - buf should always be 26 bytes long */
145 1   ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

147 1   (void) EDMDispatch_logent(
                       __FILE__,__LINE__, LOG_INFO, MESSAGE_SHUTDOWN, 0,
148 1                  "Shutting down at %s due to signal %d", ctimebuf,
                                                              sigval);


150 1   /*
151 1    * Remove our lock file.
152 1    */
153 1   (void) EslDestroyPidFile(PIDPATH);

155 1   exit(sigval);

157 } /* End of kill_handler() */
```





```
159 /*****************************************************************************
                                                                          ********
160  * Function Name:
161  *       display_usage
162  *
163  * Simply displays the usage
164  *
165  * Call Arguments:
166  *       Program name
167  *
168  * Error Outputs and Side Effects:
169  *       Prints usage.
170  *
171  * Special Considerations:
172  *       None.
173  *
174  *****************************************************************************
                                                                          ********
175  */
176 static void
177 display_usage (IN char *progname)
178 1 {
179 1   /* Print out usage stmt.  */

181 1   fprintf (stderr,"Usage: %s [-d]\n", progname);
182 1   fprintf (
      stderr,"-d keep the daemon from forking so debugging is easier\n");

184 } /* end display_usage () */
```

```
187   /*************************************************************************
                                                                        ********
188   **
189   ** Routine: daemon_catch_interrupts
190   **
191   ** Inputs:      None
192   **
193   ** Outputs:     None
194   **
195   ** Return Codes:
196   **              None
197   **
198   ** Purpose:     Sets up signals for service. On NT we will have to
199   **              consider what OS constructs to replace signals with.
200   **              In this case we are catching SIGTERM, SIGINT, and
201   **              SIGQUIT and ignoring anything else.
202   **
203   ** Intended caller:  internal only.
204   **
205   *************************************************************************
                                                                        ********
206   */
207   void daemon_catch_interrupts()
208 1 {
209 1 struct sigaction         sactions;       /* Signal actions */

211 1          ZERO( sactions );

213 1         /*
214 1          * Set an empty list so we can set signals we want to handle
215 1          */
216 1          (void) sigemptyset( &sactions.sa_mask );

218 1         /*
219 1          * Add signals that we want to handle
220 1          */
221 1          (void) sigaddset( &sactions.sa_mask, SIGTERM );
222 1          (void) sigaddset( &sactions.sa_mask, SIGINT );
223 1          (void) sigaddset( &sactions.sa_mask, SIGQUIT );

225 1          /* Setup the signal handler. */
226 1          sactions.sa_handler = kill_handler;

228 1         /*
229 1          * Assign handler to each signal we are interested in.
230 1          */
231 1          (void) sigaction( SIGTERM, &sactions, NULL );
232 1          (void) sigaction( SIGINT,  &sactions, NULL );
233 1          (void) sigaction( SIGQUIT, &sactions, NULL );

235 1          /*
236 1          * Setup mask so we can specify what signals we will ignore.
237 1          */
238 1          (void) sigfillset( &sactions.sa_mask );

240 1         /*
241 1          * We want to ignore everything except those we have set up
242 1          * above so remove those from the list.
243 1          */
244 1          (void) sigdelset( &sactions.sa_mask, SIGTERM );
245 1          (void) sigdelset( &sactions.sa_mask, SIGINT );
246 1          (void) sigdelset( &sactions.sa_mask, SIGQUIT );

248 1         /*
```





```
249 1          * Set the mask. Since no other threads have been started,
250 1          * all threads will get this mask.
251 1          */
252 1          (void) thr_sigsetmask( SIG_SETMASK, &sactions.sa_mask, NULL );
253   }
```

```
256  /*****************************************************************************
                                                                          ********
257  **
258  ** Routine: daemon_check_proper_ID
259  **
260  ** Inputs:       None
261  **
262  ** Outputs:      None
263  **
264  ** Return Codes:
265  **               exits with an error when the user is not root
266  **
267  ** Purpose:      Checks user's ID and determines if the user is allowed
268  **               to execute service.
                                  If there are no constraints then this
269  **               function may be blank.
270  **
271  ** Intended caller:  internal only.
272  **
273  *****************************************************************************
                                                                          ********
274  */

276  void daemon_check_proper_ID()
277 1 {
278 1           /*
279 1           ** Check for root
280 1           */

282 1           if (geteuid() != E_ROOTUID)
283 2           {
284 2             (void) EDMDispatch_logent(
                          __FILE__,__LINE__, LOG_ERR, DAEMON_NOTSUPERUSER, 0,
285 2                         "Must be run as superuser, uid was %d",
286 2                         geteuid());
287 2             exit(1);
288 1           }
289  }
```





```
291  /*****************************************************************************
                                                                          ********
292  **
293  ** Routine: parse_commandline
294  **
295  ** Inputs:       argc, argv (command line arguments)
296  **
297  ** Outputs:      None
298  **
299  ** Return Codes:
300  **               exits with an error when the user types a bad argument
301  **
302  ** Purpose:      Parses command line arguments and sets flags. If there
303  **               are no flags to be set then this function may be empty.
304  **
305  ** Intended caller:  internal only.
306  **
307  *****************************************************************************
                                                                          ********
308  */

310  void parse_commandline(int argc, char *argv[])
311 1 {
312 1 int                         opt;            /* Process options    */

314 1           commandlineargs = argv;

316 1           while ((opt = getopt(argc,argv,"dD")) != EOF )
317 2           {
318 2             switch(opt)
319 3             {
320 3               case 'd':
321 3               case 'D':
322 3                    G_debug = TRUE;
323 3                    break;
324 3               default:
325 3                    (void) display_usage( argv[0] );
326 3                    exit(1);

328 2             }
329 1           }
330  }
```

```
332  /*****************************************************************************
                                                                          ********
333  **
334  ** Routine: daemon_initialize_logging
335  **
336  ** Inputs:      None
337  **
338  ** Outputs:     None
339  **
340  ** Return Codes:
341  **              None
342  **
343  ** Purpose:     Do whatever it takes to initialize logging.
                                                                  In the near
344  **              future this may involve doing something with catalogs
                                                                         or
345  **              calling higher level logging functions which
                                                                  encapsulate
346  **              these things.
347  **
348  ** Intended caller:  internal only.
349  **
350  *****************************************************************************
                                                                          ********
351  */

353  void
354  daemon_initialize_logging()
355  1 {
356  1         /* Pass in argv[0], the program name */
357  1         (void) esl_log_init(commandlineargs[0]);
358    }
```





```
360  /*****************************************************************************
                                                                          ********
361  **
362  ** Routine: daemon_become_daemon
363  **
364  ** Inputs:      None
365  **
366  ** Outputs:     None
367  **
368  ** Return Codes:
369  **              exits with an error code if initialization fails
370  **
371  ** Purpose:     This function is for doing the forking etc. under UNIX.
372  **              It is unknown what will be necessary under NT.
373  **
374  ** Intended caller:  internal only.
375  **
376  *****************************************************************************
                                                                          ********
377  */

379  void
380  daemon_become_daemon()
381  1 {
382  1          char *ptr;
383  1         int  ret = 0;

385  1         /*
386  1          * Strip the path from the program name so we can use it
387  1          * elsewhere.
388  1          */
389  1         ptr = strrchr(commandlineargs[0], '/');
390  1         if (ptr == NULL)
391  1             ptr = commandlineargs[0];
392  1         else
393  1             ptr++;

395  1         /* Change directory to a process specific core directory */
396  1         ret = esl_coredir_setup(ptr);
397  1         if (ret != 0)
398  2         {
399  2              (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
400  2                                 MESSAGE_ERR_IN_ESL_COREDIR, 0,
401  2                                 "esl_coredir_setup failed: <%d>",
402  2                                 errno);
403  2             exit(1);
404  1         }

406  1          /*
407  1          **  This is now esl functionality.
                                      This code does everything necessary
408  1          **  to make this a "real" daemon by detaching from the
                                                                    terminal
409  1          **  changing the process group, closing stdout, stderr, stdin,
                                                                          ...
410  1          */

412  1         if (G_debug == FALSE)
413  2         {
414  2             ret = esl_daemon_startup();
415  2              if (ret != 0)
416  3             {
417  3                 fprintf(
                          stderr, "%s: Failed to initialize as daemon.\n",
418  3                           commandlineargs[0]);
```

```
419 3                   exit(1);
420 2               }
421 1           }
422     }
```





```
424     /*****************************************************************************
                                                                              ********
425     **
426     ** Routine: rpc_init
427     **
428     ** Inputs:      None
429     **
430     ** Outputs:     None
431     **
432     ** Return Codes:
433     **              exits with an error code if initialization fails
434     **
435     ** Purpose:     This function is for doing RPC initialization.
436     **              For the most part it involves calling the csc routines.
437     **              This is pretty standard between UNIX and NT.
438     **
439     ** Intended caller:  internal only.
440     **
441     *****************************************************************************
                                                                              ********

442     */

444     void rpc_init()
445 1   {
446 1   error_status_t          status;
                                        /* error status (nbase.h) */

447 1   char   *ebuff;

449 1   /*
450 1   ** This is here because of HP which may or may not define timeval.
451 1   ** May be removed when esl_timeval is ported to clients
452 1   */
453 1   #ifdef _STRUCT_TIMEVAL
454 1           struct timeval  sleep_interval = {5,0};
                                        /* 5 second sleep interval */

455 1   #else
456 1           struct timespec sleep_interval = {5,0};
                                        /* 5 second sleep interval */

457 1   #endif

459 1           /* Setup the interface specification for RPC */
460 1           SERVER_IFSPEC(if_spec);

462 1           /*
463 1           * Login as SERVER_PRINCIPAL.  The context of the process
464 1           * will be set to this principal.
465 1           *
466 1           * This process will keep trying to login to DCE if the
                                                                    registery
467 1           * server is unavailable.
                                Note that under SUN RPC this is a no-op.
468 1           */
469 1           while (TRUE)
470 2           {
471 2                   (void) csc_server_login(SERVER_PRINCIPAL,
472 2                                           SERVER_KEYTAB, &status);

474 2                   /* If we succeeded, then exit this loop.  */
475 2                   if ( status == error_status_ok )
476 3                   {
477 3                           break;
478 2                   }
479 2                   else    /* Print error message if appropriate. */
480 3                   {
481 3                    ebuff = (char *) csc_get_error( status );
```

```
483  3                  (void) EDMDispatch_logent(
                                        __FILE__,__LINE__,LOG_ERR,
484  3                                  MESSAGE_NO_LOGIN, 0,
485  3                                  "CSC_SERVER_LOGIN failed: <%d>
                                                              %s",
486  3                                  status, (
                                       ebuff ? ebuff : "Unknown error"));
487  2                  }

489  2                  /* If the failure was due to unavailable client,
490  2                  * pause and then try again.
491  2                  */
492  2                  if (status == sec_rgy_server_unavailable)
493  3                  {
494  3                          /*
495  3                           * uses sleep when SUNRPC, otherwise uses
496  3                           * pthread call to delay for the specified
497  3                           * time
498  3                           */
499  3                            CSC_SLEEP(sleep_interval);
500  3                            continue;
501  2                  }

503  2                  /* If we got here, we had a unexpected failure.  */
504  2                  (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
505  2                                  MESSAGE_NO_LOGIN, 0,
506  2                                 "The service cannot log in as
                                                              required");


508  2                  exit(1);
509  1          }

511  1          /*
512  1          ** We need to initialize the authorization module before we
                                                                      do
513  1          ** a listen.
514  1          */
515  1          (void)csc_authorization_init(&status);

517  1          if ( status != error_status_ok )
518  2          {
519  2           ebuff = (char *) csc_get_error( status );

521  2            (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
522  2                           MESSAGE_NOAUTHORIZATION, 0,
523  2                           "CSC_AUTHORIZATION_INIT failed: <%d> %s",
524  2                            status, (
                                      ebuff ? ebuff : "Unknown error") );
525  2            exit(1);
526  1          }


529  1          (void) csc_register_server_interface(   &if_spec,
530  1                                                  SERVER_ANNOTATION,
531  1                                                  &status);

533  1          if ( status != error_status_ok )
534  2          {
535  2           ebuff = (char *) csc_get_error( status );

537  2            (void) EDMDispatch_logent( __FILE__,__LINE__,LOG_ERR,
538  2                            MESSAGE_CANNOTREGISTER, 0,
539  2                           "CSC_REGISTER_SERVER_INTERFACE failed:
                                                              <%d> %s",
540  2                            status, (
```





```
                                      ebuff ? ebuff : "Unknown error") );
541  2            exit(1);
542  1          }
543     }
```

```
545  /*****************************************************************
                                                                ********
546  **
547  ** Routine: rpc_run
548  **
549  ** Inputs:       None
550  **
551  ** Outputs:      None
552  **
553  ** Return Codes:
554  **               None
555  **
556  ** Purpose:      This function is for running the RPC listen.
557  **               This is pretty standard between UNIX and NT.
558  **
559  ** Intended caller:  internal only.
560  **
561  *****************************************************************
                                                                ********
562  */

564  void rpc_run()
565 1 {
566 1 error_status_t            status;
                                        /* error status (nbase.h) */
567 1 char  *ebuff;

569 1          /* listen for RPC calls forever. */
570 1          (void) csc_server_listen(
                              rpc_c_listen_max_calls_default, &status );

572 1         ebuff = (char *) csc_get_error( status );

574 1          /* We don't expect to get here. */
575 1          (void) EDMDispatch_logent(
                      __FILE__,__LINE__,LOG_ERR, MESSAGE_SERVERLISTEN, 0,
576 1                              "CSC_SERVER_LISTEN failed: <%d> %s",
577 1                               status, (
                                        ebuff ? ebuff : "Unknown error") );

578  }
```





```
580  /*****************************************************************
                                                                ********
581  **
582  ** Routine: daemon_specific_initialization
583  **
584  ** Inputs:       None
585  **
586  ** Outputs:      None
587  **
588  ** Return Codes:
589  **               None
590  **
591  ** Purpose:      Do whatever makes this daemon special.
                                                    In some cases you
592  **               may want to start a thread or open a socket.
                                                      Do that here.
593  **
594  ** Intended caller:  internal only.
595  **
596  *****************************************************************
                                                                ********
597  */

599  void
600  daemon_specific_initialization()
601 1 {
602 1 error_status_t            status;
                                        /* error status (nbase.h) */
603 1 int                       ret;
604 1 pthread_t                 mantid;
605 1 pthread_t                 cleantid;
606 1 time_t    current_time;
607 1 char      *ctimebuf;
608 1 struct rlimit rlp;

610 1         /* Create a file and lock it so we don't start multiple
611 1          ** daemons.  Exit if there is another copy of us running.
612 1          ** The CreatePidFile call already logs errors so just exit.
613 1          */
614 1         if (EslCreatePidFile(PIDPATH))
615 2         {
616 2             exit(1);
617 1         }

619 1          /* Find out what time it is */
620 1          (void) time(&current_time);

622 1          ctimebuf = ctime(&current_time);

624 1          /* Overlay newline with null - buf should always be 26 bytes
                                                                  long */
625 1          ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

627 1          /* Log startup message */
628 1          (void) EDMDispatch_logent(
                          __FILE__,__LINE__, LOG_INFO, MESSAGE_STARTUP, 0,
629 1                            "Restore service %s starting up at %s",
                                                  commandlineargs[0],
630 1                            ctimebuf );

632 1         /* set the open files limit very large */
633 1         rlp.rlim_max = FD_SETSIZE;
634 1         rlp.rlim_cur = FD_SETSIZE;

636 1         setrlimit(RLIMIT_NOFILE, &rlp);
```

```
638  1          getrlimit(RLIMIT_NOFILE, &rlp);

640  1          (void) EDMDispatch_logent(
                          __FILE__,__LINE__, LOG_INFO, MESSAGE_STARTUP, 0,
641  1                       "Service allows %d open files",
                                                        rlp.rlim_max );

643  1          /* Initialize service launcher */
644  1          ret = EDMDDSvcInit();

646  1          if (ret != 0)
647  2          {
648  2               (void) EDMDispatch_logent(
                                        __FILE__,__LINE__, LOG_INFO, 0, 0,
649  2                            "Service Launcher failed returning - %d",
                                                                    ret);
650  2               exit(1);
651  1          }

653  1          /*
654  1           * Start the other threads in the daemon. The main thread
655  1           * becomes the RPC thread. BAMDataManage is the entry point
656  1           * for the data collection thread. BAMDataCleanup is the
657  1           * entry point for the data expiration thread.
658  1            */
659  1          /*pthread_create(&mantid, NULL, DispDaemon_ccr, NULL); */
660  1          pthread_create(&mantid, NULL, DispDaemon_ccw, NULL);
661  1          pthread_create(&cleantid, NULL, DispatchBackground, NULL);
662  1          rpc_init();
663  1          rpc_run();
664     }
```





```
666  /*****************************************************************************
                                                                          ********
667  **
668  ** Routine: daemon_cleanup
669  **
670  ** Inputs:       None
671  **
672  ** Outputs:      None
673  **
674  ** Return Codes:
675  **               None
676  **
677  ** Purpose:      Call function which will clean up daemon properly.
678  **
679  ** Intended caller:  internal only.
680  **
681  *****************************************************************************
                                                                          ********
682  */

684  void
685  daemon_cleanup()
686  1 {
687  1          kill_handler( 0 );
688  }
```

Thu Dec 27 11:27:11 2007 Function Index Page 2

Thu Dec 27 11:27:11 2007 Function Index Page 1

Thu Dec 27 11:27:11 2007 Function Index Page 2

```
1   /*
2   ** Copyright 1996,1997 EMC Corporation
3   */


6   /* EDMDispatchService.c
7    *
8    *
9    * Mission Statement:  RPC entry points.
10   *
11   * Primary Data Acted On:
12   *
13   * Compile-Time Options:
14   *
15   * Basic idea here:
16   */

18  #if !defined(lint)
19  static char     RCS_id [] = "@(#)$RCSfile: EDMDispatchService.c,v $ "
20                              "$Revision: 1.0 $ "
21                              "$Date: 1997/02/06 20:49:15 $" ;
22  #endif

24  #include <esl/c_portable.h>
25  #include <esl/inout.h>

27  #include <logging/logging.h>
28  #include <csc/cscomm.h>

30  #include <restore/csc_EDMDispatch.h>
31  #include <restore/dispatch_daemon.h>

33  #include <EDMDispatchLog.h>
34  #include <EDMDispatchSession.h>

36  /*
37   * These are all the rpc entry points for the dispatch daemon.
38   * The dispatch daemon is multi-threaded and it is the main thread
39   * which handles all incoming RPC. ONC RPC is single threaded
40   * so each call blocks other RPC calls. This provides us some
41   * safety in the way we handle our data and limits our exposure
42   * to unexpected multithreading problems.
43   */
44  static void FreeSessionInfo(SessionInfo *);

46  /*****************************************************************************
                                                                        ********
47  **
48  ** Routine:  dd_initialize_1
49  **
50  ** Inputs:   DD_initialize_args * - args for the restore initialize
                                                                           call
51  **
52  ** Outputs:  None
53  **
54  ** Return Codes:
55  **           DD_initialize_result * - result of init function call
56  **
57  ** Purpose:  Function to create a restore session.
58  **
59  ** Intended caller:  Internal Only.
60  *****************************************************************************
                                                                        ********
61  */

63  DD_initialize_result *
```





```
64  dd_initialize_1_svc(
                    IN DD_initialize_args *arg, IN struct svc_req *req )
65  1 {
66  1     static DD_initialize_result argzz;

68  1     InitializeSession(arg, req, &argzz);

70  1     return &argzz;
71    }
```

```
73  /*****************************************************************************
                                                                        ********
74  **
75  ** Routine:  dd_getservicestatus_1
76  **
77  ** Inputs:   DD_getservicestatus_args * - args for the
                                                    getservicestatus call
78  **
79  ** Outputs:  None
80  **
81  ** Return Codes:
82  **           DD_getservicestatus_result * - result of status function
                                                                      call
83  **
84  ** Purpose:  Function to poll for status on a session.
85  **
86  ** Intended caller:  Internal Only.
87  *****************************************************************************
                                                                        ********
88  */

90  DD_getservicestatus_result *
91  dd_getservicestatus_1_svc(
                IN DD_getservicestatus_args *arg, IN struct svc_req *req )
92  1 {
93  1     static DD_getservicestatus_result argzz;

95  1     GetDispatchStatus(arg, &argzz);

97  1     return &argzz;
98    }
```





```
100 /*****************************************************************************
                                                                        ********
101 **
102 ** Routine:  dd_getsessioninfo_1
103 **
104 ** Inputs:   DD_getservicestatus_args * - args for the getsessioninfo
                                                                      call
105 **
106 ** Outputs:  None
107 **
108 ** Return Codes:
109 **           SessionBlock * - result of session info call
110 **
111 ** Purpose:  Function to get information on all sessions.
112 **
113 ** Intended caller:  Internal Only.
114 *****************************************************************************
                                                                        ********
115 */

117 SessionBlock *
118 dd_getsessioninfo_1_svc(
              IN DD_getservicestatus_args *arg, IN struct svc_req *req )
119 1 {
120 1     static SessionBlock argzz;
121 1     static boolean_ty first = TRUE;

123 1     if (first)
124 2     {
125 2         memset(&argzz, 0, sizeof(argzz));
126 2         first = FALSE;
127 1     }
128 1     else
129 2     {
130 2         FreeSessionInfo(argzz.sess);
131 2         argzz.sess = NULL;
132 1     }

134 1     GetDispatchInfo(arg, &argzz);

136 1     return &argzz;
137   }
```

```
139  /*************************************************************************
                                                                        ********
140  **
141  ** Routine:  FreeSessionInfo
142  **
143  ** Inputs:   SessionInfo * - arg to free
144  **
145  ** Outputs:  None
146  **
147  ** Return Codes:
148  **           None
149  **
150  ** Purpose:  Function to free all SessionInfo structures in a list.
151  **
152  ** Intended caller:  Internal Only.
153  *************************************************************************
                                                                     *********/

155  static void FreeSessionInfo(SessionInfo *sess)
156 1{
157 1      if (sess == NULL)
158 1          return;

160 1      if (sess -> next != NULL)
161 1          FreeSessionInfo(sess -> next);

163 1      free(sess);
164  }
```